Lecture 2: Asymptotic Analysis, Recurrences

Michael Dinitz

September 2, 2021

601.433/633 Introduction to Algorithms

Things I Forget on Tuesday

Level of Formality:

- Part of mathematical maturity is knowing when to be formal, when not necessary
- Rule of thumb: Be formal for important parts
  - Problem 1 is *about* asymptotic notation. Be formal!
  - Problem 2 is *about* recurrences. Can be a little less formal with asymptotic notation.
- Lectures:
  - I tend to go fast, not be super formal. But I expect you to be formal in homeworks (unless stated otherwise)

Handwriting:

- I have bad handwriting. If something's not clear, ask!

Today

Should be review, some might be new.
See math background in CLRS

Asymptotics: $\mathbf{O}(\cdot)$, $\boldsymbol{\Omega}(\cdot)$, and $\boldsymbol{\Theta}(\cdot)$ notation.

- Should know from Data Structures. We'll be a bit more formal.
- Intuitively: hide constants and lower order terms, since we only care what happen "at scale" (asymptotically)

Recurrences: How to solve recurrence relations.

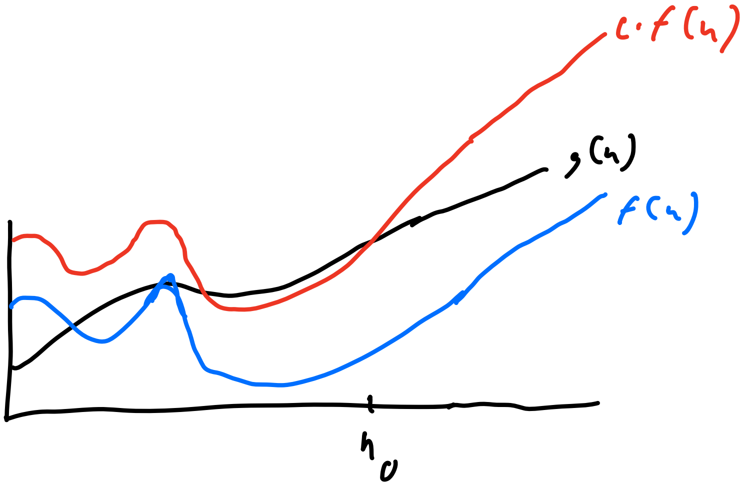
- Should know from Discrete Math.

Asymptotic Notation

$O(\cdot)$

Definition

$\mathbf{g(n) \in O(f(n))}$ if there exist constants $\mathbf{c, n_0 > 0}$ such that $\mathbf{g(n) \le c \cdot f(n)}$ for all $\mathbf{n > n_0}$.



$O(\cdot)$

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \leq c \cdot f(n)$ for all $n > n_0$.

Technically $O(f(n))$ is a set.

Abuse notation: "$g(n)$ is $O(f(n))$" or $g(n) = O(f(n))$. $g(n) \leq O(f(n))$

$O(\cdot)$

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \le c \cdot f(n)$ for all $n > n_0$.

Technically $O(f(n))$ is a set.
Abuse notation: "$g(n)$ is $O(f(n))$" or $g(n) = O(f(n))$.

Examples:

- $2n^2 + 27 = O(n^2)$: set $n_0 = 6$ and $c = 3$
- $2n^2 + 27 = O(n^3)$: same values, or $n_0 = 4$ and $c = 1$
- $n^3 + 2000n^2 + 2000n = O(n^3)$: set $n_0 = 10000$ and $c = 2$

$O(\cdot)$

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \le c \cdot f(n)$ for all $n > n_0$.

Technically $O(f(n))$ is a set.
Abuse notation: "$g(n)$ is $O(f(n))$" or $g(n) = O(f(n))$.

Examples:

- $2n^2 + 27 = O(n^2)$: set $n_0 = 6$ and $c = 3$
- $2n^2 + 27 = O(n^3)$: same values, or $n_0 = 4$ and $c = 1$
- $n^3 + 2000n^2 + 2000n = O(n^3)$: set $n_0 = 10000$ and $c = 2$

About *functions* not algorithms!
Expresses an *upper* bound

Example

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \le c \cdot f(n)$ for all $n > n_0$.

Theorem

$2n^2 + 27 = O(n^2)$

Example

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \leq c \cdot f(n)$ for all $n > n_0$.

Theorem

$2n^2 + 27 = O(n^2)$

Proof.

Set $c = 3$. Suppose $2n^2 + 27 > cn^2 = 3n^2$

Example

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \leq c \cdot f(n)$ for all $n > n_0$.

Theorem

$2n^2 + 27 = O(n^2)$

Proof.

Set $c = 3$. Suppose $2n^2 + 27 > cn^2 = 3n^2$

$\implies n^2 < 27$

Example

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \leq c \cdot f(n)$ for all $n > n_0$.

Theorem

$2n^2 + 27 = O(n^2)$

Proof.

Set $c = 3$. Suppose $2n^2 + 27 > cn^2 = 3n^2$

$\implies n^2 < 27 \implies n < 6$

Example

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \leq c \cdot f(n)$ for all $n > n_0$.

Theorem

$2n^2 + 27 = O(n^2)$

Proof.

Set $c = 3$. Suppose $2n^2 + 27 > cn^2 = 3n^2$

$\implies n^2 < 27 \implies n < 6$

$\implies 2n^2 + 27 \leq 3n^2$ for all $n \geq 6$.

Example

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \leq c \cdot f(n)$ for all $n > n_0$.

Theorem

$2n^2 + 27 = O(n^2)$

Proof.

Set $c = 3$. Suppose $2n^2 + 27 > cn^2 = 3n^2$
$\implies n^2 < 27 \implies n < 6$
$\implies 2n^2 + 27 \leq 3n^2$ for all $n \geq 6$.
Set $n_0 = 6$. Then $2n^2 + 27 \leq cn^2$ for all $n > n_0$. $\square$

Example

Definition

$g(n) \in O(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \leq c \cdot f(n)$ for all $n > n_0$.

Theorem

$2n^2 + 27 = O(n^2)$

Proof.

Set $c = 3$. Suppose $2n^2 + 27 > cn^2 = 3n^2$

$\implies n^2 < 27 \implies n < 6$

$\implies 2n^2 + 27 \leq 3n^2$ for all $n \geq 6$.

Set $n_0 = 6$. Then $2n^2 + 27 \leq cn^2$ for all $n > n_0$. $\square$

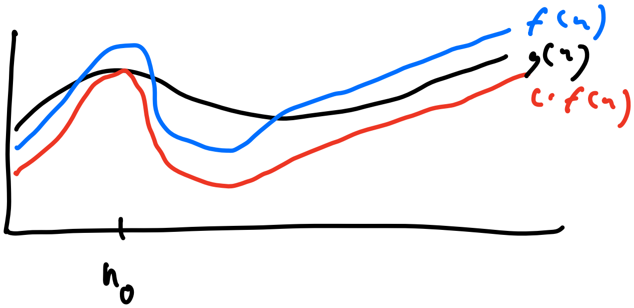
Many other ways to prove this!

$\Omega(\cdot)$

Counterpart to $\mathbf{O}(\cdot)$: *lower* bound rather than upper bound.

Definition

$\mathbf{g(n)} \in \mathbf{\Omega(f(n))}$ if there exist constants $\mathbf{c, n_0 > 0}$ such that $\mathbf{g(n) \geq c \cdot f(n)}$ for all $\mathbf{n > n_0}$.



$\Omega(\cdot)$

Counterpart to $O(\cdot)$: *lower* bound rather than upper bound.

Definition

$g(n) \in \Omega(f(n))$ if there exist constants $c, n_0 > 0$ such that $g(n) \geq c \cdot f(n)$ for all $n > n_0$.

Examples:

- $2n^2 + 27 = \Omega(n^2)$: set $n_0 = 1$ and $c = 1$
- $2n^2 + 27 = \Omega(n)$: set $n_0 = 1$ and $c = 1$
- $\frac{1}{100}n^3 - 1000n^2 = \Omega(n^3)$: set $n_0 = 1000000$ and $c = 1/1000$

$\Theta(\cdot)$

Combination of $O(\cdot)$ and $\Omega(\cdot)$.

Definition

$g(n) \in \Theta(f(n))$ if $g(n) \in O(f(n))$ and $g(n) \in \Omega(f(n))$.

Note: constants n_0, c can be different in the proofs for $O(f(n))$ and $\Omega(f(n))$

$\Theta(\cdot)$

Combination of $O(\cdot)$ and $\Omega(\cdot)$.

Definition

$g(n) \in \Theta(f(n))$ if $g(n) \in O(f(n))$ and $g(n) \in \Omega(f(n))$.

Note: constants n_0, c can be different in the proofs for $O(f(n))$ and $\Omega(f(n))$

Equivalent:

Definition

$g(n) \in \Theta(f(n))$ if there are constants $c_1, c_2, n_0 > 0$ such that $c_1 f(n) \leq g(n) \leq c_2 f(n)$ for all $n > n_0$.

Both lower bound and upper bound, so asymptotic equality.

Little notation

Strict versions of $\mathbf{O}$ and $\mathbf{\Omega}$:

Definition

$g(n) \in o(f(n))$ if for every constant $c > 0$ there exists a constant $n_0 > 0$ such that $g(n) < c \cdot f(n)$ for all $n > n_0$.

Definition

$g(n) \in \omega(f(n))$ if for every constant $c > 0$ there exists a constant $n_0 > 0$ such that $g(n) > c \cdot f(n)$ for all $n > n_0$.

Examples:

- $2n^2 + 27 = o(n^2 \log n)$
- $2n^2 + 27 = \omega(n)$

$2n^2 \neq o(n^2)$ $2n^2+27 = O(n^2 \log n)$

Recurrence Relations

Sorting

Many algorithms recursive so running time naturally a recurrence relation (Karatsuba, Strassen).

Sorting

Many algorithms recursive so running time naturally a recurrence relation (Karatsuba, Strassen).

Sorting:

- Selection Sort

Sorting

Many algorithms recursive so running time naturally a recurrence relation (Karatsuba, Strassen).

Sorting:

- Selection Sort
  - Find smallest unsorted element, put it just after sorted elements. Repeat.

Sorting

Many algorithms recursive so running time naturally a recurrence relation (Karatsuba, Strassen).

Sorting:

- Selection Sort
  - Find smallest unsorted element, put it just after sorted elements. Repeat.
  - Running time: Takes $O(n)$ time to find smallest unsorted element, decreases remaining unsorted by 1.

    $\implies T(n) = T(n-1) + cn$

    def of T

Sorting

Many algorithms recursive so running time naturally a recurrence relation (Karatsuba, Strassen).

Sorting:

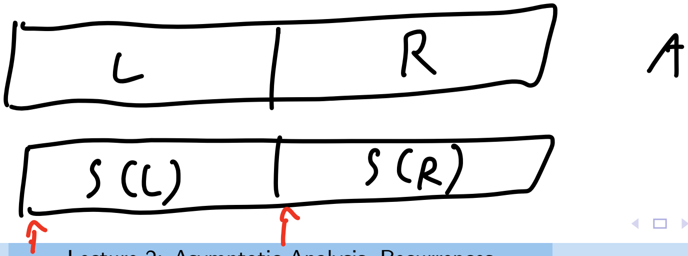
- Selection Sort
  - Find smallest unsorted element, put it just after sorted elements. Repeat.
  - Running time: Takes $\mathbf{O(n)}$ time to find smallest unsorted element, decreases remaining unsorted by $\mathbf{1}$.
    $\implies \mathbf{T(n) = T(n-1) + cn}$
- Mergesort

Sorting

Many algorithms recursive so running time naturally a recurrence relation (Karatsuba, Strassen).

Sorting:

- Selection Sort
  - Find smallest unsorted element, put it just after sorted elements. Repeat.
  - Running time: Takes $\mathbf{O(n)}$ time to find smallest unsorted element, decreases remaining unsorted by $\mathbf{1}$.
    $\implies \mathbf{T(n) = T(n-1) + cn}$
- Mergesort
  - Split array into left and right halves. Recursively sort each half, then merge.



Sorting

Many algorithms recursive so running time naturally a recurrence relation (Karatsuba, Strassen).

Sorting:

- Selection Sort
  - Find smallest unsorted element, put it just after sorted elements. Repeat.
  - Running time: Takes $O(n)$ time to find smallest unsorted element, decreases remaining unsorted by 1.

    $$\implies T(n) = T(n-1) + cn$$
- Mergesort
  - Split array into left and right halves. Recursively sort each half, then merge.
  - Running time: Merging takes $O(n)$ time. Two recursive calls on half the size.

    $$\implies T(n) = \underbrace{2T(n/2)}_{\text{recursive calls}} + \underset{\text{merging}}{cn}$$

Sorting

Many algorithms recursive so running time naturally a recurrence relation (Karatsuba, Strassen).

Sorting:

- Selection Sort
  - Find smallest unsorted element, put it just after sorted elements. Repeat.
  - Running time: Takes $O(n)$ time to find smallest unsorted element, decreases remaining unsorted by 1.
    $\implies T(n) = T(n-1) + cn$
- Mergesort
  - Split array into left and right halves. Recursively sort each half, then merge.
  - Running time: Merging takes $O(n)$ time. Two recursive calls on half the size.
    $\implies T(n) = 2T(n/2) + cn$

Also need base case. For algorithms, constant size input takes constant time.
$\implies T(n) \leq c$ for all $n \leq n_0$, for some constants $n_0, c > 0$.

Guess and Check

$$T(n) = 3T(n/3) + n \qquad\qquad T(1) = 1$$

Guess and Check

$$T(n) = 3T(n/3) + n \qquad\qquad T(1) = 1$$

Guess: $T(n) \leq cn$.

Guess and Check

$$T(n) = 3T(n/3) + n \qquad\qquad T(1) = 1$$

Guess: $T(n) \le cn$.

Check: assume true for $n' < n$, prove true for n (induction).

Guess and Check

$$T(n) = 3T(n/3) + n \qquad\qquad T(1) = 1$$

Guess: $T(n) \le cn$.

Check: assume true for $n' < n$, prove true for n (induction).

$$T(n) = 3T(n/3) + n \le 3cn/3 + n = (c+1)n$$

def induction algebra

induction: $T(n/3) \le c \cdot \frac{n}{3}$

Guess and Check

$$T(n) = 3T(n/3) + n \qquad\qquad T(1) = 1$$

Guess: $T(n) \le cn$.

Check: assume true for $n' < n$, prove true for n (induction).

$$T(n) = 3T(n/3) + n \le 3cn/3 + n = (c+1)n$$

Failure! Wanted $T(n) \le cn$, got $T(n) \le (c+1)n$. Guess was wrong.

Guess and Check

$$T(n) = 3T(n/3) + n \qquad\qquad T(1) = 1$$

Guess: $T(n) \leq cn$.

Check: assume true for $n' < n$, prove true for n (induction).

$$T(n) = 3T(n/3) + n \leq 3cn/3 + n = (c+1)n$$

Failure! Wanted $T(n) \leq cn$, got $T(n) \leq (c+1)n$. Guess was wrong.

Better guess? What goes up by 1 when n goes up by a factor of 3?

Guess and Check

$$T(n) = 3T(n/3) + n \qquad\qquad T(1) = 1$$

Guess: $T(n) \leq cn$.

Check: assume true for $n' < n$, prove true for n (induction).

$$T(n) = 3T(n/3) + n \leq 3cn/3 + n = (c+1)n$$

Failure! Wanted $T(n) \leq cn$, got $T(n) \leq (c+1)n$. Guess was wrong.

Better guess? What goes up by 1 when n goes up by a factor of 3? $\log_3 n$

Guess and Check

$$T(n) = 3T(n/3) + n \qquad\qquad T(1) = 1$$

Guess: $T(n) \le cn$.

Check: assume true for $n' < n$, prove true for n (induction).

$$T(n) = 3T(n/3) + n \le 3cn/3 + n = (c+1)n$$

Failure! Wanted $T(n) \le cn$, got $T(n) \le (c+1)n$. Guess was wrong.

Better guess? What goes up by 1 when n goes up by a factor of 3? $\log_3 n$

Guess: $T(n) \le n\log_3(3n)$

Check: assume true for $n' < n$, prove true for n (induction).

$$T(n) \overset{\text{def}}{=} 3T(n/3) + n \overset{\text{IH}}{\le} 3(n/3)\log_3(\overset{3\cdot\frac{n}{3}}{n}) + n = n\log_3(n) + n$$
$$= n(\log_3(n) + \log_3 3) = n\log_3(3n).$$

Unrolling

Example: selection sort. $\mathbf{T(n) = T(n-1) + cn}$

Idea: "unroll" the recurrence.

Unrolling

Example: selection sort. $\mathbf{T(n) = T(n-1) + cn}$

Idea: "unroll" the recurrence.

$$\mathbf{T(n) = cn + T(n-1)}$$

Unrolling

Example: selection sort. $\mathbf{T(n) = T(n-1) + cn}$

Idea: “unroll” the recurrence.

$$
\begin{aligned}
\mathbf{T(n)} &= \mathbf{cn + T(n-1)} \\
&= \mathbf{cn + c(n-1) + T(n-2)}
\end{aligned}
$$

def of $T(n-1)$

Unrolling

Example: selection sort. $T(n) = T(n-1) + cn$

Idea: "unroll" the recurrence.

$$\begin{aligned} T(n) &= cn + T(n-1) \\ &= cn + c(n-1) + T(n-2) \\ &= cn + c(n-1) + c(n-2) + T(n-3) \end{aligned}$$

Unrolling

Example: selection sort. $T(n) = T(n-1) + cn$

Idea: “unroll” the recurrence.

$$\begin{aligned} T(n) &= cn + T(n-1) \\ &= cn + c(n-1) + T(n-2) \\ &= cn + c(n-1) + c(n-2) + T(n-3) \\ &\vdots \end{aligned}$$

Unrolling

Example: selection sort. $T(n) = T(n-1) + cn$

Idea: "unroll" the recurrence.

$$\begin{aligned} T(n) &= cn + T(n-1) \\ &= cn + c(n-1) + T(n-2) \\ &= cn + c(n-1) + c(n-2) + T(n-3) \\ &\;\;\vdots \\ &= cn + c(n-1) + c(n-2) + \cdots + c \end{aligned}$$

Unrolling

Example: selection sort. $T(n) = T(n-1) + cn$

Idea: "unroll" the recurrence.

$$\begin{aligned} T(n) &= cn + T(n-1) \\ &= cn + c(n-1) + T(n-2) \\ &= cn + c(n-1) + c(n-2) + T(n-3) \\ &\vdots \\ &= cn + c(n-1) + c(n-2) + \cdots + c \end{aligned}$$

n terms, each of which at most cn $\implies$ $T(n) \le cn^2 = O(n^2)$

Unrolling

Example: selection sort. $\mathbf{T(n) = T(n-1) + cn}$

Idea: "unroll" the recurrence.

$$\begin{aligned}
\mathbf{T(n)} &= \mathbf{cn + T(n-1)} \\
&= \mathbf{cn + c(n-1) + T(n-2)} \\
&= \mathbf{cn + c(n-1) + c(n-2) + T(n-3)} \\
&\vdots \\
&= \mathbf{cn + c(n-1) + c(n-2) + \cdots + c}
\end{aligned}$$

$\mathbf{n}$ terms, each of which at most $\mathbf{cn}$ $\implies$ $\mathbf{T(n) \le cn^2 = O(n^2)}$

At least $\mathbf{n/2}$ terms which are at least $\mathbf{cn/2}$ $\implies$ $\mathbf{T(n) \ge c\frac{n^2}{4} = \Omega(n^2)}$

Unrolling

Example: selection sort. $\mathbf{T(n) = T(n-1) + cn}$

Idea: "unroll" the recurrence.

$$
\begin{aligned}
\mathbf{T(n)} &= \mathbf{cn + T(n-1)} \\
&= \mathbf{cn + c(n-1) + T(n-2)} \\
&= \mathbf{cn + c(n-1) + c(n-2) + T(n-3)} \\
&\vdots \\
&= \mathbf{cn + c(n-1) + c(n-2) + \cdots + c}
\end{aligned}
$$

$cn + c(n-1) + c(n-2) + \dots$

$+ 4c + 3c + 2c + c$

n terms, each of which at most **cn** $\implies \mathbf{T(n) \le cn^2 = O(n^2)}$

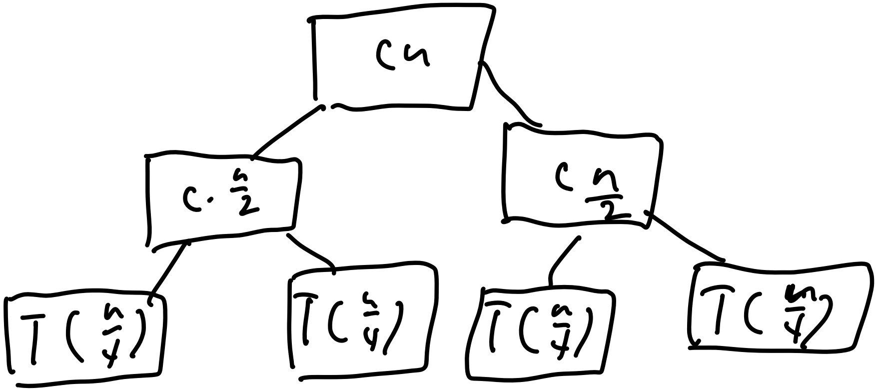
At least **n/2** terms which are at least **cn/2** $\implies \mathbf{T(n) \ge c\frac{n^2}{4} = \Omega(n^2)}$

$\implies \mathbf{T(n) = \Theta(n^2)}$.

Recursion Tree: Mergesort

Generalizes unrolling: draw out full tree of "recursive calls".

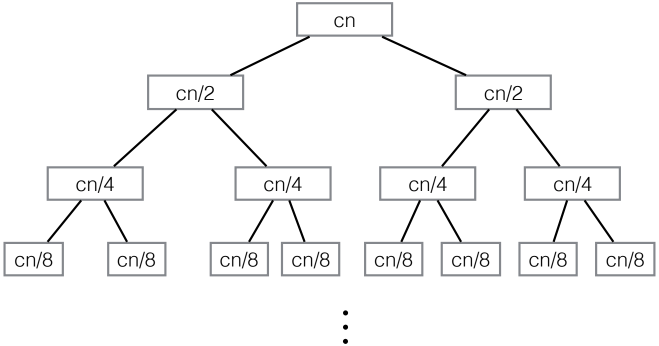
Mergesort: $\mathbf{T(n) = 2T(n/2) + cn}$.



Recursion Tree: Mergesort

Generalizes unrolling: draw out full tree of “recursive calls”.

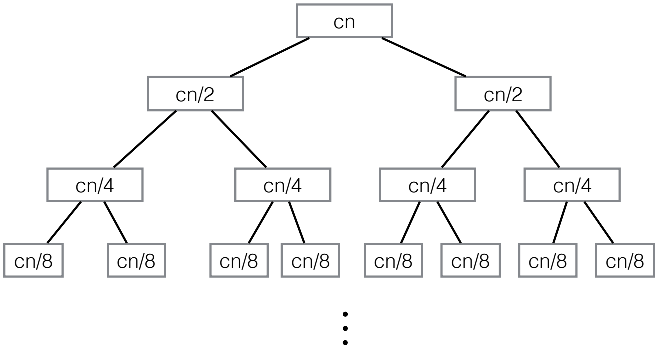
Mergesort: $\mathbf{T(n) = 2T(n/2) + cn}$.



Recursion Tree: Mergesort

Generalizes unrolling: draw out full tree of "recursive calls".

Mergesort: $\mathbf{T(n) = 2T(n/2) + cn}$.

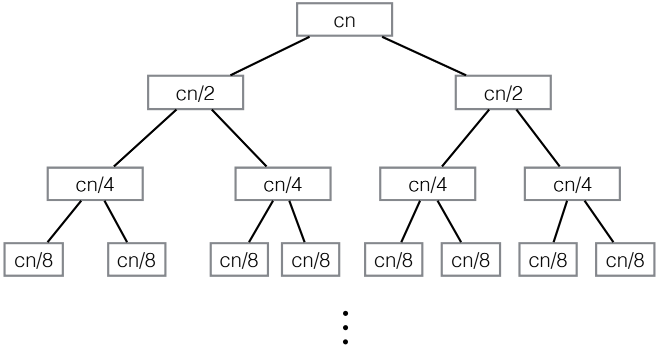


levels:

Recursion Tree: Mergesort

Generalizes unrolling: draw out full tree of "recursive calls".

Mergesort: $T(n) = 2T(n/2) + cn$.

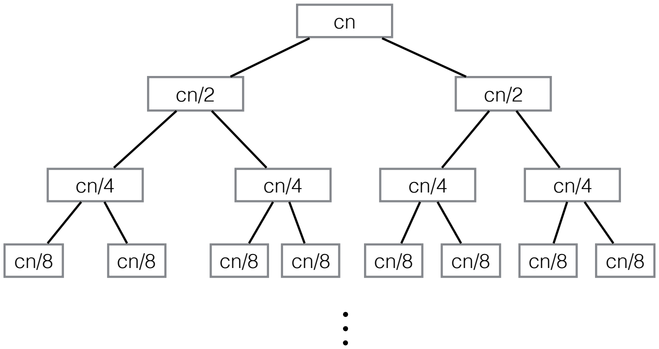


\# levels: $\log_2 n$

Recursion Tree: Mergesort

Generalizes unrolling: draw out full tree of "recursive calls".

Mergesort: $\mathbf{T(n) = 2T(n/2) + cn}$.



cn

cn

cn

cn

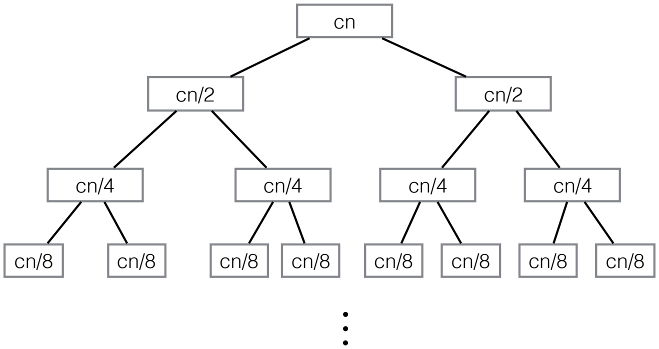
levels: $\mathbf{\log_2 n}$

Contribution of level $\mathbf{i}$:

Recursion Tree: Mergesort

Generalizes unrolling: draw out full tree of "recursive calls".

Mergesort: $\mathbf{T(n) = 2T(n/2) + cn}$.



level 0

1

2

3

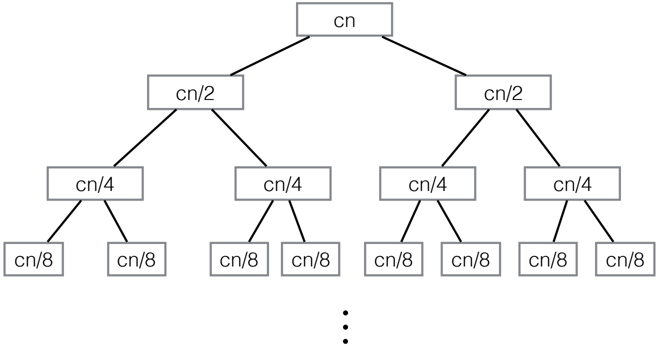
\# levels: $\mathbf{\log_2 n}$

Contribution of level $\mathbf{i}$: $\mathbf{2^{i-1} cn/2^{i-1} = cn}$

Recursion Tree: Mergesort

Generalizes unrolling: draw out full tree of “recursive calls”.

Mergesort: $\mathbf{T(n) = 2T(n/2) + cn}$.



levels: $\mathbf{\log_2 n}$

Contribution of level **i**: $\mathbf{2^{i-1} cn/2^{i-1} = cn}$

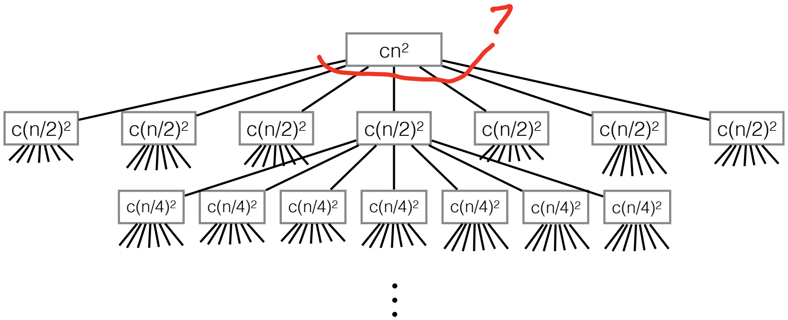
$\implies \mathbf{T(n) = \Theta(n \log n)}$

Recursion Tree: Strassen

$$T(n) = 7T(n/2) + cn^2$$

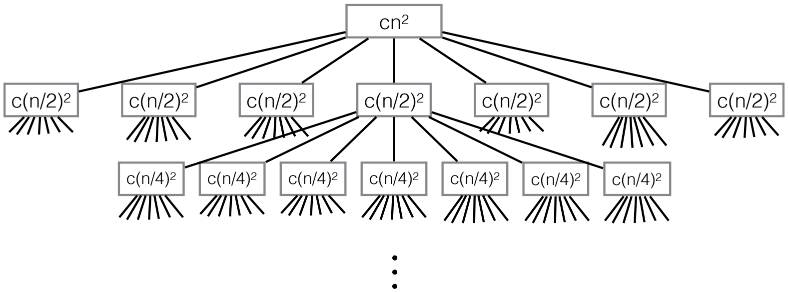
Recursion Tree: Strassen

$$T(n) = 7T(n/2) + cn^2$$



Recursion Tree: Strassen

$$T(n) = 7T(n/2) + cn^2$$



level 1

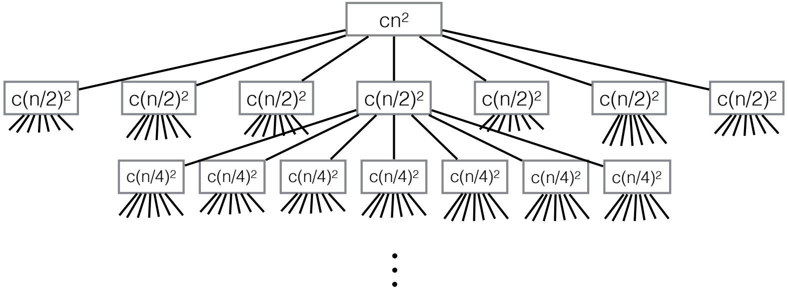
2

3

Level **i**: $7^{i-1}c(n/2^{i-1})^2 = (7/4)^{i-1}cn^2$

Recursion Tree: Strassen

$$T(n) = 7T(n/2) + cn^2$$

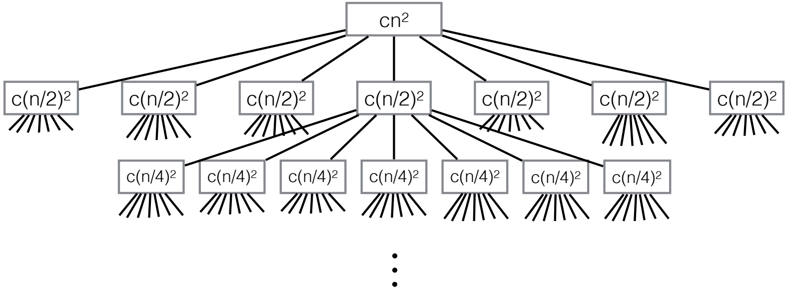


Level **i**: $7^{i-1}c(n/2^{i-1})^2 = (7/4)^{i-1}cn^2$

Total:
$$T(n) = \sum_{i=1}^{\log n+1} \left(\frac{7}{4}\right)^{i-1} cn^2 = cn^2 \sum_{i=1}^{\log n+1} \left(\frac{7}{4}\right)^{i-1}$$

Recursion Tree: Strassen

$$T(n) = 7T(n/2) + cn^2$$



Level **i**: $7^{i-1}c(n/2^{i-1})^2 = (7/4)^{i-1}cn^2$

Total:

$$T(n) = \sum_{i=1}^{\log n+1} \left(\frac{7}{4}\right)^{i-1} cn^2 = cn^2 \sum_{i=1}^{\log n+1} \left(\frac{7}{4}\right)^{i-1}$$

$$\Longrightarrow T(n) = O(n^2(7/4)^{\log n}) = O(n^2 n^{\log(7/4)}) = O(n^2 n^{\log 7-2})$$

$$= O(n^{\log 7})$$

Master Theorem

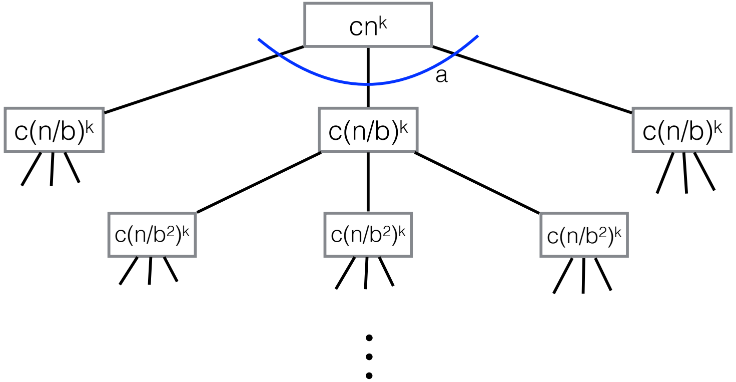
$$T(n) = aT(n/b) + cn^k \qquad T(1) = c$$

a, b, c, k constants with $a \geq 1$, $b > 1$, $c > 0$, and $k \geq 0$

Master Theorem

$$T(n) = aT(n/b) + cn^k \qquad T(1) = c$$

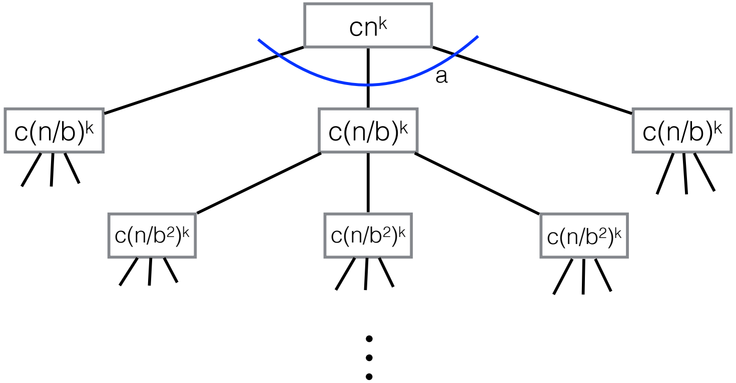
a, b, c, k constants with $a \geq 1$, $b > 1$, $c > 0$, and $k \geq 0$



Master Theorem

$$T(n) = aT(n/b) + cn^k \qquad T(1) = c$$

$\mathbf{a, b, c, k}$ constants with $\mathbf{a \geq 1}$, $\mathbf{b > 1}$, $\mathbf{c > 0}$, and $\mathbf{k \geq 0}$

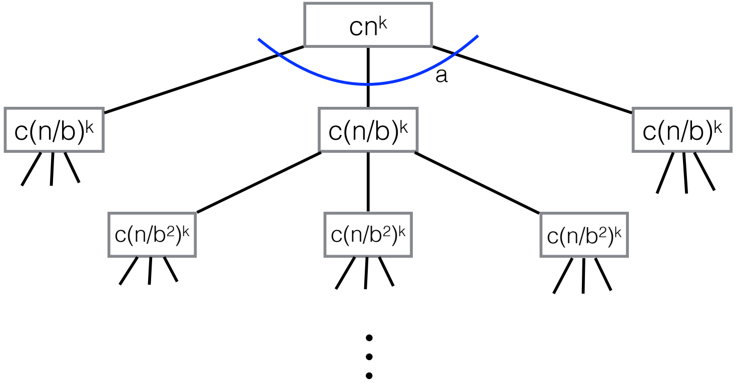


\# levels: $\mathbf{\log_b n + 1}$

Master Theorem

$$T(n) = aT(n/b) + cn^k \qquad\qquad T(1) = c$$

a, b, c, k constants with $a \geq 1$, $b > 1$, $c > 0$, and $k \geq 0$



\# levels: $\log_b n + 1$

Level i: $a^{i-1}c(n/b^{i-1})^k = cn^k(a/b^k)^{i-1}$

Master Theorem II

Let $\alpha = (a/b^k)$

$\implies T(n) = cn^k \sum_{i=1}^{\log_b n+1} (a/b^k)^{i-1} = cn^k \sum_{i=1}^{\log_b n+1} \alpha^{i-1}$

Master Theorem II

Let $\alpha = (a/b^k)$

$\implies T(n) = cn^k \sum_{i=1}^{\log_b n+1} (a/b^k)^{i-1} = cn^k \sum_{i=1}^{\log_b n+1} \alpha^{i-1}$

- Case 1: $\alpha = 1$. All levels the same. $T(n) = cn^k \sum_{i=1}^{\log_b n+1} 1 = \Theta(n^k \log n)$

Master Theorem II

Let $\alpha = (a/b^k)$

$$\implies T(n) = cn^k \sum_{i=1}^{\log_b n+1} (a/b^k)^{i-1} = cn^k \sum_{i=1}^{\log_b n+1} \alpha^{i-1}$$

- Case 1: $\alpha = 1$. All levels the same. $T(n) = cn^k \sum_{i=1}^{\log_b n+1} 1 = \Theta(n^k \log n)$
- Case 2: $\alpha < 1$. Dominated by top level.

Master Theorem II

Let $\alpha = (a/b^k)$

$\implies T(n) = cn^k \sum_{i=1}^{\log_b n+1} (a/b^k)^{i-1} = cn^k \sum_{i=1}^{\log_b n+1} \alpha^{i-1}$

- Case 1: $\alpha = 1$. All levels the same. $T(n) = cn^k \sum_{i=1}^{\log_b n+1} 1 = \Theta(n^k \log n)$
- Case 2: $\alpha < 1$. Dominated by top level.
  $\implies \sum_{i=1}^{\log_b n+1} \alpha^{i-1} \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \frac{1}{1-\alpha}$.
  $\implies T(n) = O(n^k)$

Master Theorem II

Let $\alpha = (a/b^k)$

$\implies T(n) = cn^k \sum_{i=1}^{\log_b n+1} (a/b^k)^{i-1} = cn^k \sum_{i=1}^{\log_b n+1} \alpha^{i-1}$

- Case 1: $\alpha = 1$. All levels the same. $T(n) = cn^k \sum_{i=1}^{\log_b n+1} 1 = \Theta(n^k \log n)$
- Case 2: $\alpha < 1$. Dominated by top level.

  $\implies \sum_{i=1}^{\log_b n+1} \alpha^{i-1} \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \frac{1}{1-\alpha}$.

  $\implies T(n) = O(n^k)$

  $T(n) \geq cn^k \implies T(n) = \Omega(n^k)$

Master Theorem II

Let $\alpha = (a/b^k)$

$\implies T(n) = cn^k \sum_{i=1}^{\log_b n+1} (a/b^k)^{i-1} = cn^k \sum_{i=1}^{\log_b n+1} \alpha^{i-1}$

- Case 1: $\alpha = 1$. All levels the same. $T(n) = cn^k \sum_{i=1}^{\log_b n+1} 1 = \Theta(n^k \log n)$
- Case 2: $\alpha < 1$. Dominated by top level.

  $\implies \sum_{i=1}^{\log_b n+1} \alpha^{i-1} \le \sum_{i=1}^{\infty} \alpha^{i-1} = \frac{1}{1-\alpha}$.

  $\implies T(n) = O(n^k)$

  $T(n) \ge cn^k \implies T(n) = \Omega(n^k) \implies T(n) = \Theta(n^k)$

Master Theorem II

Let $\alpha = (a/b^k)$

$$\implies T(n) = cn^k \sum_{i=1}^{\log_b n+1} (a/b^k)^{i-1} = cn^k \sum_{i=1}^{\log_b n+1} \alpha^{i-1}$$

- Case 1: $\alpha = 1$. All levels the same. $T(n) = cn^k \sum_{i=1}^{\log_b n+1} 1 = \Theta(n^k \log n)$
- Case 2: $\alpha < 1$. Dominated by top level.

  $\implies \sum_{i=1}^{\log_b n+1} \alpha^{i-1} \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \frac{1}{1-\alpha}$.

  $\implies T(n) = O(n^k)$

  $T(n) \geq cn^k \implies T(n) = \Omega(n^k) \implies T(n) = \Theta(n^k)$
- Case 3: $\alpha > 1$. Dominated by bottom level

Master Theorem II

Let $\alpha = (a/b^k)$

$\implies T(n) = cn^k \sum_{i=1}^{\log_b n+1} (a/b^k)^{i-1} = cn^k \sum_{i=1}^{\log_b n+1} \alpha^{i-1}$

- Case 1: $\alpha = 1$. All levels the same. $T(n) = cn^k \sum_{i=1}^{\log_b n+1} 1 = \Theta(n^k \log n)$
- Case 2: $\alpha < 1$. Dominated by top level.

  $\implies \sum_{i=1}^{\log_b n+1} \alpha^{i-1} \le \sum_{i=1}^{\infty} \alpha^{i-1} = \frac{1}{1-\alpha}$.

  $\implies T(n) = O(n^k)$

  $T(n) \ge cn^k \implies T(n) = \Omega(n^k) \implies T(n) = \Theta(n^k)$
- Case 3: $\alpha > 1$. Dominated by bottom level

  $$\implies \sum_{i=1}^{\log_b n+1} \alpha^{i-1} = \alpha^{\log_b n} \sum_{i=1}^{\log_b n+1} \left(\frac{1}{\alpha}\right)^{i-1} \le \alpha^{\log_b n} \frac{1}{1-(1/\alpha)}$$

  $$= O(\alpha^{\log_b n})$$

Master Theorem II

Let $\alpha = (a/b^k)$

$\implies T(n) = cn^k \sum_{i=1}^{\log_b n+1} (a/b^k)^{i-1} = cn^k \sum_{i=1}^{\log_b n+1} \alpha^{i-1}$

- Case 1: $\alpha = 1$. All levels the same. $T(n) = cn^k \sum_{i=1}^{\log_b n+1} 1 = \Theta(n^k \log n)$
- Case 2: $\alpha < 1$. Dominated by top level.

  $\implies \sum_{i=1}^{\log_b n+1} \alpha^{i-1} \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \frac{1}{1-\alpha}$.

  $\implies T(n) = O(n^k)$

  $T(n) \geq cn^k \implies T(n) = \Omega(n^k) \implies T(n) = \Theta(n^k)$
- Case 3: $\alpha > 1$. Dominated by bottom level

$$\implies \sum_{i=1}^{\log_b n+1} \alpha^{i-1} = \alpha^{\log_b n} \sum_{i=1}^{\log_b n+1} \left(\frac{1}{\alpha}\right)^{i-1} \leq \alpha^{\log_b n} \frac{1}{1-(1/\alpha)}$$

$$= O(\alpha^{\log_b n})$$

$$\implies T(n) = \Theta(n^k \alpha^{\log_b n}) = \Theta(n^k (a/b^k)^{\log_b n}) = \Theta(a^{\log_b n})$$

$$= \Theta(n^{\log_b a})$$

Master Theorem III

Theorem ("Master Theorem")

The recurrence

$$T(n) = aT(n/b) + cn^k \qquad\qquad T(1) = c$$

where a, b, c, *and* k *are constants with* $a \geq 1$, $b > 1$, $c > 0$, *and* $k \geq 0$, *is equal to*

$$T(n) = \Theta(n^k) \text{ if } a < b^k,$$
$$T(n) = \Theta(n^k \log n) \text{ if } a = b^k,$$
$$T(n) = \Theta(n^{\log_b a}) \text{ if } a > b^k.$$